Launch Commit Criteria Monitoring Agent

Glenn S. Semmel
SPS Branch, YA-D8
NASA
KSC, FL 32899
Glenn.S.Semmel@nasa.gov

Steven R. Davis
SPS Branch, YA-D8
NASA
KSC, FL 32899
Steve.Davis@nasa.gov

Kurt W. Leucht
SPS Branch, YA-D8
NASA
KSC, FL 32899
Kurt.Leucht@nasa.gov

Dan A. Rowe
SPS Branch, YA-D8
NASA
KSC, FL 32899
Daniel.A.Rowe@nasa.gov

Andrew O. Kelly
PH-G2
NASA
KSC, FL 32899
Andrew.O.Kelly@nasa.gov

Ladislau Bölöni
Dept. Elec. and Comp. Eng.
University of Central Florida
Orlando, FL 32816
lboloni@cpe.ucf.edu

ABSTRACT

The Spaceport Processing Systems Branch at NASA Kennedy Space Center has developed and deployed a software agent to monitor the Space Shuttle's ground processing telemetry stream. The application, the Launch Commit Criteria Monitoring Agent, increases situational awareness for system and hardware engineers during Shuttle launch countdown. The agent provides autonomous monitoring of the telemetry stream, automatically alerts system engineers when predefined criteria have been met, identifies limit warnings and violations of launch commit criteria, aids Shuttle engineers through troubleshooting procedures, and provides additional insight to verify appropriate troubleshooting of problems by contractors. The agent has successfully detected launch commit criteria warnings and violations on a simulated playback data stream. Efficiency and safety are improved through increased automation.

Categories and Subject Descriptors

I.2.1 [**Applications and Expert Systems**]: Industrial automation; I.4 [**Pattern Recognition**]: Applications

General Terms

design

Keywords

agent, expert system, rule-based programming

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

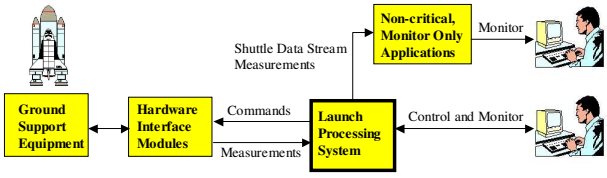

Figure 1: Ground Control and Monitoring at NASA KSC

1. INTRODUCTION

This paper describes a software agent [1] that is used for processing Space Shuttle telemetry data and notifying system engineers of warnings and violations. After describing the problem and objectives, the environment, interfaces, application description, and extension of the agent for future uses will be presented.

1.1 Background

NASA Kennedy Space Center (KSC) is responsible for prelaunch ground checkout of the Space Shuttle. The Launch Processing System (LPS) at KSC provides facilities for NASA Shuttle system engineers, contractors, and test conductors to command, control, and monitor space vehicle systems from the start of Shuttle interface testing through various phases including terminal countdown, launch, abort, safing, and scrub turnaround.

LPS continually monitors the Shuttle and its ground equipment including environmental controls and hardware that loads propellants. Consoles with vehicle responsibilities communicate information directly to and from the Shuttle computer systems. Consoles with ground support equipment responsibility communicate information to and from the hardware interface modules which are connected to the numerous ground support systems. See Figure 1. Each module is capable of interfacing to approximately 240 sensors or controls. Overall, some 50,000 temperatures, pressures, flow rates, liquid levels, turbine speeds, voltages, currents, valve positions, switch positions, and many other parameters must be controlled and monitored.

For over 25 years, engineers have used LPS to verify Space Shuttle flight readiness and to control launch countdown. LPS has performed superbly well. Recently, much of the LPS hardware was

[1]A demonstration of this system will be available to be shown at the conference.

Subsystem	Number LCCs	Number Measurements
APU/HYD	50	252
ECLSS	29	136
PRSD	15	113
OMS	18	434

Table 1: Number of Measurements for Various Shuttle Subsystems

upgraded assuring its continuance for many more years. However, the system architecture was not changed and the software remains basically the same. As a result, the level of situational awareness has not increased proportionally to what would otherwise be possible with more modern software technologies.

After the Shuttle Columbia disaster on February 1, 2003, the Columbia Accident Investigation Board [15] proposed recommendations to improve safety from both an organizational and technical perspective. The Board indicated the need to "[adopt] and maintain a Shuttle flight schedule that is consistent with available resources." Also, both management and engineering support staff must maintain an awareness of anomalies and those must not be lost "as engineering risk analyses [move] through the process." Given two tragic losses of a crew and Shuttle, today NASA engineers have an even greater pressure to be more vigilant in identifying problems. Anomalies must be detected and reported to prevent problems with Shuttle subsystems, countdown, and launch. The aging LPS hardware has limited resources and precludes the level of automation and notification warranted by this domain.

1.2 Problem Description

During launch countdown, NASA Shuttle engineers are required to monitor shuttle data for violations of the launch commit criteria (LCC) and to verify that the contractors troubleshoot problems correctly. When a violation is recognized by the system engineers it is reported to the NASA Test Director. The problem report, or call, includes a description of the problem, the criticality, whether a hold is requested, and whether a preplanned troubleshooting procedure exists.

The Shuttle is composed of many subsystems (e.g. Main Propulsion, Hydraulics). Each of those subsystems has a team of engineers responsible for troubleshooting problems for that respective system during a launch countdown. Many systems have a large number of measurements with associated LCC limits and a large number of LCC requirements. Table 1 shows four representative Shuttle subsystems and their corresponding number of LCCs and measurements. As illustrated, hundreds of measurements must be monitored just for this small set of subsystems.

Shuttle Engineers must monitor for many types of limit violations ranging from simple high and low limit boundaries to much more complex first order logic expressions. Each team has its own tools for identifying LCC violations. Many of these tools use the LPS software and simply change the color of the displayed data and/or present a text message to the user or set off an audible alarm. Troubleshooting may require other displays such as plots and troubleshooting flowcharts. Valuable time is spent locating these procedures and locating the data that supports them.

Given the complexity of the logical expressions that specify limit violations, the large number of limits, and the need to find supporting data quickly, Shuttle engineers sought an advisory tool to provide more insight and situational awareness during launch countdown. In the latter half of 2003, a software tool was proposed to address these needs during launch countdown. The tool, called the Launch Commit Criteria Monitoring Agent (LCCMA), complements LPS and is capable of autonomously and continuously monitoring Shuttle telemetry data. LCCMA automatically alerts NASA Shuttle engineers when predefined criteria (e.g. limit violations, warnings) have been met and guides the engineers through troubleshooting procedures.

1.3 Objectives

LCCMA acts as a software agent for the NASA engineer. For this discussion, an agent is defined as rule-based, autonomous software that reacts to its environment and communicates results to a human, a NASA engineer in this usage. Agents have been extensively researched [24, 21]. Agents standards [10] and frameworks [1, 16] have also been developed.

The primary objectives for LCCMA include:

- Monitor Space Shuttle telemetry ground data.
- Allow a NASA engineer to specify rules to be applied to Space Shuttle telemetry ground data.
- Display a visual indication of violated LCCs.
- Display a text message of the LCC violation call.
- Display troubleshooting steps from preplanned procedures.

LCCMA does not send any commands and is used for advisory purposes only. A future release of LCCMA will include an interactive troubleshooting display that reads the data stream and accepts user inputs to direct diagnostic troubleshooting.

2. ENVIRONMENT AND INTERFACES

2.1 Shuttle Data Stream

Data processed by LPS is distributed on a local area network. As shown in Figure 1, the distributed data is known as the Shuttle Data Stream (SDS) [17] and contains real-time vehicle and ground processing data. Thousands of telemetry measurements are published in the SDS and are used by monitor-only applications such as LCCMA. The SDS contains multiple types and subtypes of measurements including discretes (i.e. boolean measurements), analogs (i.e. floating point measurements), and digital patterns (i.e. integer measurements).

2.2 LCCMA Context Diagram

Figure 2 shows the context diagram for LCCMA. The agent process, represented in the middle circle, communicates with various sources and data stores. A measurement database is used to decode the SDS into usable measurements. The SDS source broadcasts measurements as data packets over local area networks. LCCMA monitors this stream for measurement violations and warnings specified by the Shuttle engineers. The Troubleshooting Procedures source represents html or pdf files containing the troubleshooting steps, often in flowchart format. LCCMA sends limit violations to the NASA Engineer via the Status Board Display. The Rules data store represents the scripts and knowledge base that defines the rules for the limit violations.

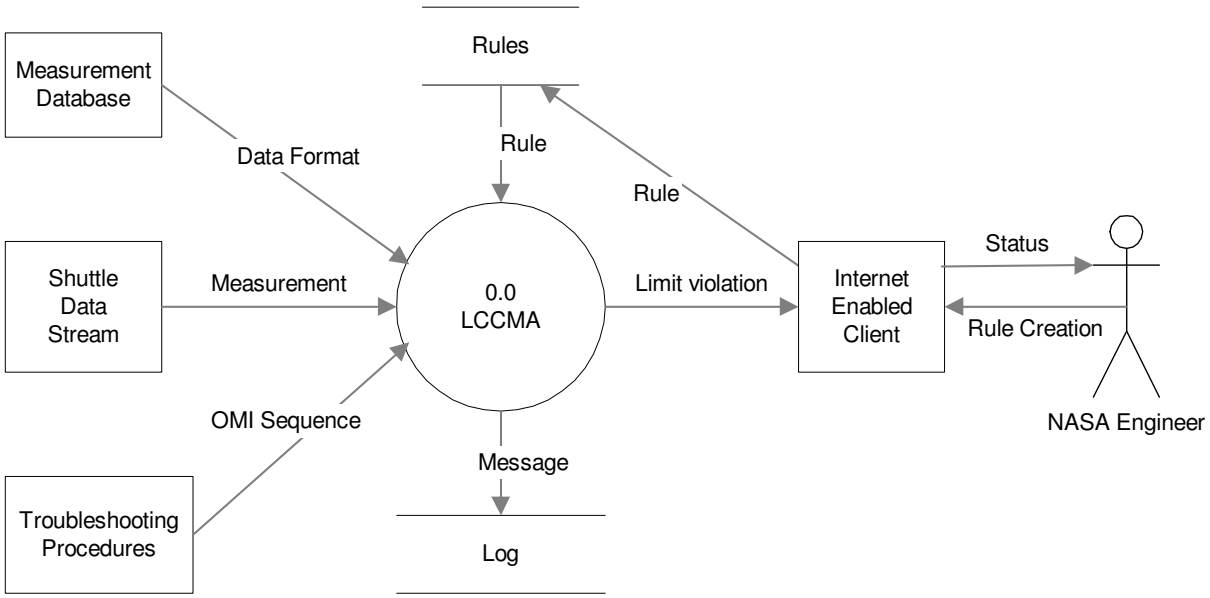

Figure 2: LCCMA Context Diagram

3. APPLICATION DESCRIPTION

3.1 Languages and Agent Tools Used in Application

The Java Expert System Shell (Jess) [13] was selected as the agent's rule engine. Jess was developed and supported by another government agency, Sandia National Labs. As such, our development team and customer have full usage of the tool via government licensing without any fees. This includes access to all the Jess source code.

Jess' forward chaining reasoning system was modeled after production systems such as OPS5 [3] and CLIPS [25]. It contains highly efficient and sophisticated pattern matching based on the Rete algorithm [12]. This enables its inference engine to process many rules and data rapidly. The engine repeatedly processes through a match-select-act cycle. As a production system, its consequents can be actions. A conflict resolution strategy determines the precedence of rule firings.

Jess' predicate logic lends itself to capturing and specifying the heuristics and engineering rules of this spaceport domain. The declarative paradigm of this rule-based agent application also makes it highly modular and scalable to span multiple subsystems of the Shuttle. Jess also includes a fourth generation scripting language and interactive command line which are very conducive for prototyping and testing.

Jess is written entirely in Java and has access to the full Java application programming interface within its scripting language. It provides standard control flow constructs and supports variables, strings, objects, and function calls. Jess automatically converts between its own types and Java types insulating the developer from manually performing the conversions. Its use as a Java library made Jess' selection more appealing since Java supports multiple platforms with its "write once, run anywhere" paradigm. Beyond that, the need for LCCMA to support web enabled clients also made Java a natural fit given its origins and strong support for developing Internet based applications.

3.2 Design

Java classes were developed to parse and decode the data stream and represent measurements as facts in Jess' working memory. To interface Jess' rule engine with the SDS, each data measurement is modeled and implemented as a Java bean [23]. Java beans provide a component architecture to enable easier integration of applications. A property change notification mechanism is supported that allows one object to become a registered listener of another object. The listener object will then automatically receive changes from the source object. This is also known as a publish-subscribe or observer pattern [14]. Within Jess, each Java bean corresponds to what is known as a shadow fact. A Jess shadow fact is a mirror image of a Java bean, such as a pressure measurement, within Jess' working memory. All shadow facts are registered listeners of their Java bean counterparts. Thus, whenever a measurement changes in the data stream, a property change event is automatically generated for the given measurement and its sibling shadow fact is updated in Jess' working memory. Figure 3 illustrates this path.

After a shadow fact is updated, the Jess pattern matcher will determine if the premises of any rules match the new or modified facts. Rules are compared to working memory to identify premises that are matched by the data in working memory. For LCCMA, this data represents measurements from the SDS and rules represent data monitoring criteria submitted by NASA Shuttle system engineers. Rules with matching premises are activated and placed onto an agenda. Next, the agenda is ordered according to Jess' default conflict resolution strategy. The highest priority rule is then fired and executed. This match-select-act cycle repeats until no more rules are available to fire. An action handler class was developed and is used to build and send the notification message to the Shuttle engineer whenever a rule fires.

3.3 Graphical User Interface

A graphical user interface currently exists for LCCMA called the Status Board Display. It is being upgraded and Figure 4 shows a storyboard representative of that future interface. The Status Board Display shows the health of the network connection, data stream status, countdown time, and other relevant information.

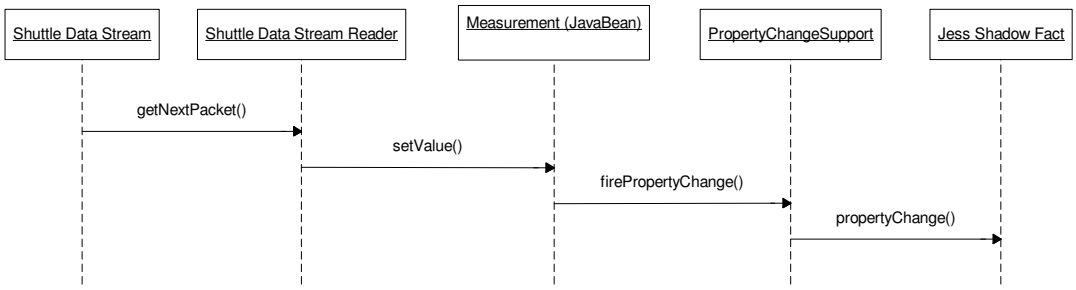

Figure 3: Sequence Diagram Illustrating Update to Jess Working Memory from Shuttle Data Stream

When LCC limits are violated, the LCC call is displayed in the text box. The user reads the text and, if there is an associated troubleshooting file, clicks the file button next to the text. This brings up a Troubleshooting Display for that particular LCC and limit. The LCC text remains bold until the Acknowledge button is pressed. Message text can be displayed with one of three icons representing a violation, warning, or informational cue.

The text messages can be read over the Operational Intercommunication System as LCC calls during the countdown. Calls will change based on what limit is violated (e.g. warning, LCC, high/low limit), the time criticality of the call, and LCC effectivity. The application aids the NASA engineer in making a Go/No-Go decision.

3.4 Execution

At startup, LCCMA connects to a single data stream based on user input and reads a rules file containing LCC violations and warning limits. Table 2 shows the conditions and actions associated with an LCC warning and violation for a hydrogen (H_2) tank in the Power Reactant Storage and Distribution (PRSD) subsystem. For instance, if either of the H_2 tank 1 pressures are above an upper limit, the agent should notify the NASA engineer by displaying the violation in red font and provide a link to the corresponding troubleshooting file (i.e. PRSD06Hi.pdf) for that violation. The troubleshooting file shows the steps necessary to be taken by the engineer when the specified limits of a given subsystem are violated.

3.5 Performance Requirements and Testing

3.5.1 Performance Characteristics of Shuttle Data Stream

At application startup, LCCMA connects to a datastream selected by the user. The datastream includes all measurements at their respective change rates. No data changes will be missing from this stream. For this discussion, only this data stream, known as FIFO, will be presented as it is the stream of choice for the customer.

The datastream averages 5 to 10 packets per second and peaks around 50 packets per second at launch. Each SDS data packet can hold up to 360 measurement changes before rolling over to another packet. This calculates to an average of 1,800 changes per second for the FIFO stream nominally, and 18,000 changes per second peak at launch. During peak data loads, the SDS is throttled at the source and does not maintain true real time updates. It may lag up to 1 minute or so, but all measurement changes are buffered and none is ever dropped from the data stream. Throttling of the data typically begins at T+1 second, that is, just after launch. Even though it is the hypothetical peak limit, 18,000 changes per second is the performance load that LCCMA is expected to meet to avoid missing a measurement change. This is referring strictly to updating 18,000 facts per second and not indicating how many rules might fire. In fact, only a small percentage of those facts is expected to result in a small percentage of the total rules to fire at any given time, even during the peak launch data rates.

The measurement data in the stream is refreshed every three minutes regardless as to whether or not it has changed. Since the stream is based on User Datagram Protocol (UDP), this results in an unreliable datagram packet service. When a packet is dropped on the network, all measurements are marked invalid and the measurements change back to valid one by one as refresh data is received until the completion of a three minute refresh cycle.

3.5.2 Performance Testing

Performance testing occurred on an Intel Pentium 4, 1.7 GHz desktop workstation with 768 MB of RAM running Microsoft Windows XP Professional. The SDS reader class in LCCMA parses the data stream and updates facts in Jess' working memory. To test the reader class, 12 high speed analog measurements were selected and instantiated as shadow facts. In the range of 18,000 (nominal) to 36,000 (peak at launch) data changes occurred every second in the test-enhanced data stream and were processed by the SDS reader class. This included various types of measurements such as discretes and analogs. 12,000 analog data changes per second were being processed into current values and updated in Jess' working memory by a property change event handler.

Rules were written for 6 of the high speed analog measurements. The other 6 measurements were still relevant to stress the SDS reader class and updating of facts. 5 of the 6 rules fired once every minute. The 6th rule fired once for every single measurement change (1,000 per sec) for two full seconds sustained out of every minute. Thus, a total of 2005 rules fired every minute, with 2000 of them firing within a 2 second period. Analog measurements have considerably more processing overhead than the discrete measurements so it was not possible to sustain thousands of rules containing analogs to fire every second without causing CPU starvation. However, the "fair test" was considered to have only a very small percentage of the measurements that are in the stream actually causing rules to fire. It was considered fair to have short bursts of high rate rule firings but not long term sustained high rate rule firings. LCCMA is not intended for users to write rules to notify them via the Status Board Display hundreds or thousands of times each second for a long and sustained period of time.

To summarize, the agent sustained the above scenario for many cycles on the test-enhanced playback file without CPU starvation and without reporting any packet losses. The CPU utilization on the development workstation was about 90% prior to launch and higher than that after T-0. It was heavily loaded, but the agent maintained

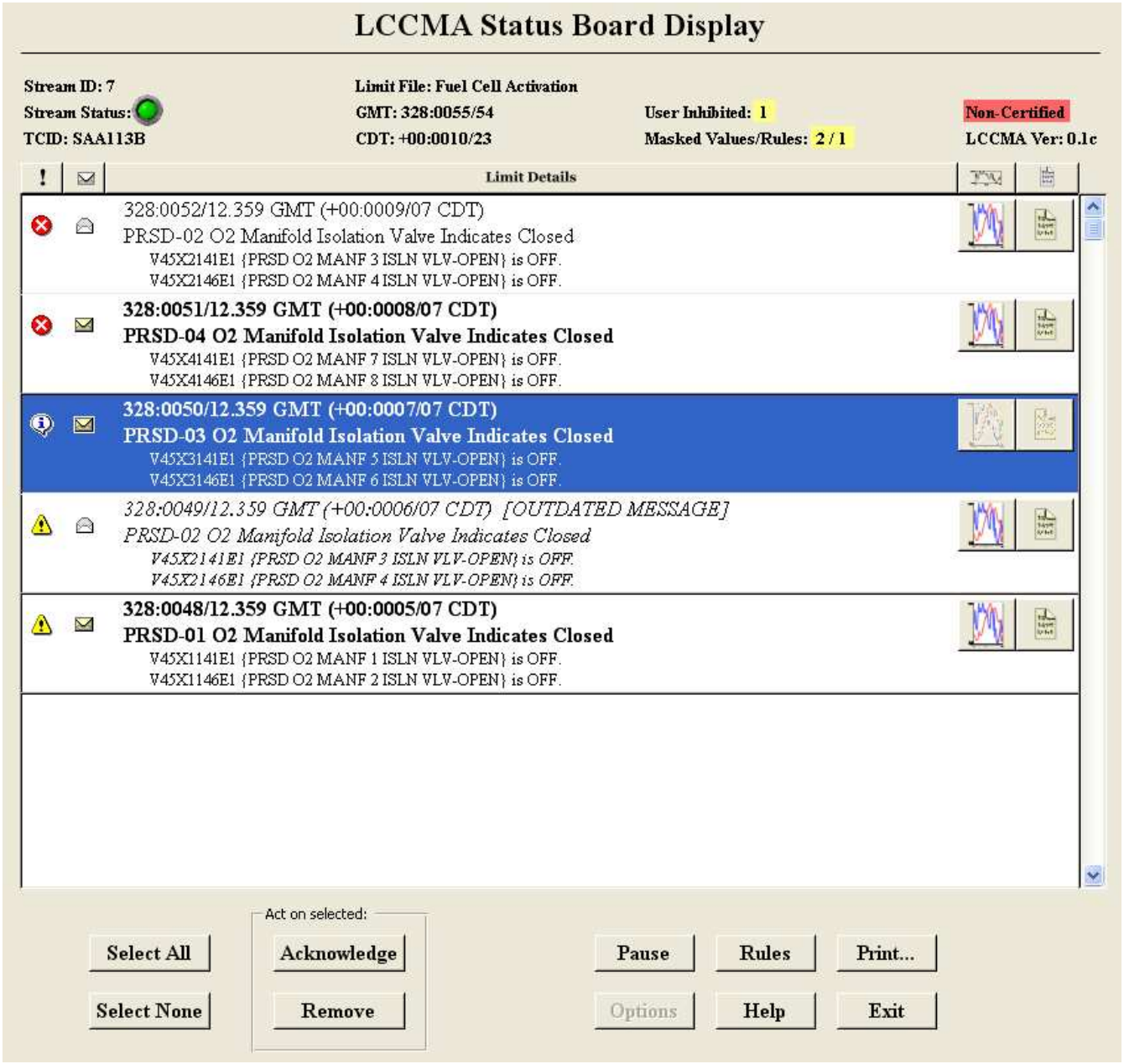

Figure 4: LCCMA Status Board Display

the pace. The agent performed well considering that the data stream was stuffed with between 1 and 2 times the hypothetical peak load of measurement changes for the performance test. The "long pole" in the process appeared to be the number of rules that actually fired every second sustained. However, even under launch conditions when a heavy data change load exists, there is not expected to be many thousands of rules firing every second. Even several hundred rules firing per minute is considered unrealistically high, but this performace test suggests the agent, LCCMA, could readily handle that load.

3.6 Deployment

LCCMA was delivered to the customer and has successfully detected LCC warnings and violations on an SDS recorded playback. It has not been used during an actual launch countdown yet since NASA has not returned to flight subsequent to the Columbia disaster. However, LCCMA's potential was already recognized by other projects at NASA KSC and it is in the process of being integrated into a larger monitoring application. For that one, thousands of NASA engineers and contractors will use LCCMA to enter not just LCC monitoring criteria, but many other types of simple and complex measurement constraints.

Condition	Description	Message	Action
(V45P2110A > 270 OR V45P2100A > 270) AND V45P2110A <= 298 AND V45P2100A <= 294	H_2 Tank 1 Pressure Warning High	Heater Control Pressure Reading [V45P2110A], Tank Pressure Reading [V45P2100A]	Display [Description], [Message] in Yellow
(V45P2110A > 298 OR V45P2100A > 294)	H_2 Tank 1 Pressure Violation High	Heater Control Pressure Reading [V45P2110A], Tank Pressure Reading [V45P2100A]	Display [Description], [Message] in Red. Open file PRSD06Hi.pdf

Table 2: Example LCC Conditions and Actions for PRSD H_2 Tank 1

3.7 Knowledge Representation

This is an actual LCCMA rule written in the Jess scripting language:

```
(defrule orbiter-cabin-o2-pressure-anomaly-rule
  "ECL-06 Emergency Condition Yellow Orbiter Cabin O2 Pressure Anomaly"

  ?activation-fact <- (activate-orbiter-cabin-o2-pressure-anomaly-rule)
  (AnalogFd (fdName "V61P2511A1") (value ?V61P2511A1_val))
  (AnalogFd (fdName "V61P2513A1") (value ?V61P2513A1_val))
  (AnalogFd (fdName "V61P2515A1") (value ?V61P2515A1_val))

  (test
    (or
      (> (abs(- ?V61P2511A1_val ?V61P2513A1_val )) 0.15 )
      (> (abs(- ?V61P2513A1_val ?V61P2515A1_val )) 0.15 )
      (> (abs(- ?V61P2511A1_val ?V61P2515A1_val )) 0.15 )
      (> (abs(- ?V61P2511A1_val 3.1)) 0.3)
      (> (abs(- ?V61P2513A1_val 3.1)) 0.3)
      (> (abs(- ?V61P2515A1_val 3.1)) 0.3)

    )
  )

  =>

  (retract ?activation-fact)
  (assert(orbiter-cabin-o2-pressure-anomaly-rule-reactivation-activate))
  (notifyActionHandler
    (create$
     "http://xb70.ksc.nasa.gov/ECL/ECL_Home/Launch.html"
     "http://xb70.ksc.nasa.gov/ECL/ECL_Home/Cabin_Leak.html"
    )
    (get-member LccmaColor message_violation)
    (create$ V61R2401A1 V61P2405A1 V61T2552A1)
  )
)
```

For this rule, the following three analog measurements are monitored: V61P2511A1, V61P2513A1, and V61P2515A1. The absolute value of the difference among pairs of these analog measurements must not exceed a given quantity. If anyone of them is exceeded, the rule will fire indicating an anomaly in the cabin oxygen pressure. Once fired, the right hand side of the rule executes. The `notifyActionHandler` call has three arguments. The first one contains two troubleshooting web page links that are made available to the NASA engineer. The second argument specifies the color of the message, a violation in this case. Finally, the third argument species the three measurements that may be plotted to investigate the anomaly further.

4. FUTURE EXPLORATION AGENTS

As indicated in the national Vision for Space Exploration [19, 20], an increased human and robotic presence will be cultivated in space, on lunar and Martian surfaces, and other destinations. Spaceports will now span from the Earth to the Moon and beyond. A new set of challenges is presented by this Exploration Vision. In particular, the need for autonomy significantly increases as people and payloads are sent greater distances from Earth.

Agents for these future applications will demand much higher degrees of autonomy than today's Shuttle agents. Few or no human experts will reside at remote lunar or Martian sites to correct problems in a timely manner. More automation will be required along with advanced diagnostics and prognostics. This requires higher levels of reasoning.

Today on Earth, system and hardware engineers along with technicians leverage multiple skills when monitoring, diagnosing, and prognosticating problems in Shuttle ground support equipment. For the Exploration Vision, the need for extending these skills to support other vehicles at remote locations from the Earth to Mars becomes essential. These skills include being rational, collaborative, goal driven, and the ability to reason over time and uncertainty. The agent discussed earlier in the paper, LCCMA, is capable of shallowing reasoning of short inference chains within the Shuttle domain. However, this existing agent can be endowed with higher levels of rationality enabling a deeper reasoning. We are investigating how to mature LCCMA into a Spaceport Exploration Agent (SEA) in support of the Exploration Vision.

SEAs will need to communicate and collaborate along multiple and lengthy logistics chains. This does not simply include agents monitoring pre-flight checkout of vehicles at a terrestrial spaceport (e.g. LCCMA monitoring Shuttle activities). Rather, SEAs will reside in multiple locations at great distances. Logistics, scheduling, and planning are just some of the activities that these agents will manage.

Within this virtual collaborative management chain, SEAs will be inundated with massive amounts of data that must be sorted and processed. It becomes necessary for them to revise their sets of beliefs as new data arrives. It is simply not enough to revise singular data points within an agent's working memory and to have an agent blindly react to those changes. Rather, an agent must possess the ability to revise previously concluded assertions based on what may be now stale data. This activity is called truth maintenance [7, 4, 11], also known as belief revision, and is particularly important when deep reasoning of long inferences is necessary. An assumption based truth maintenance system (ATMS) can reason over many contexts simultaneously. By capturing, maintaining, and deploying spaceport expertise within ATMS-enabled SEAs, the costs and manpower required to meet the Exploration Vision are reduced while safety, reliability, and availability are increased.

4.1 Benefits of Endowing Spaceport Exploration Agents with Belief Revision

SEAs enabled with belief revision will provide the following:

- SEAs will continuously monitor spaceport telemetry streams for expected and anomalous conditions during operations and launch countdowns. SEAs will analyze data from networks of sensors and draw inferences over time to deduce further action. Results are provided to humans, agents, and other subsystems which may compose an integrated health management function.
- SEAs provide an automated explanation generation facility and diagnostic capabilities. The inferences and facts that lead

to a conclusion will be available to the human expert and other agents for further processing.

- SEAs provide prognostics to predict where and when failures may occur in support equipment and what if scenarios to assess chains of events.
- If a human expert leaves the program or moves onto other opportunities, SEAs remain and can virtually mentor the human replacement leveraging its knowledge base.

4.2 Extending LCCMA with Truth Maintenance

As indicated earlier, LCCMA uses Jess as its inference engine. Jess implements a lightweight version of truth maintenance that is much simpler than a full blown ATMS. Jess' *logical* conditional element keeps track of the "here and now" for specified premises. Other rule based systems, such as Clips and Lisa [26], implement a similar level of truth maintenance.

Premises on the left hand side of a rule can be tied to assertions of facts on the right hand side via the *logical* keyword. A dependency is created between the facts of the premise and the fact of the conclusion. After the rule fires and the consequent's fact(s) is asserted, if the premise ever becomes false, the consequent's facts will be automatically retracted assuming other logical support does not exist for those facts. In contrast to Jess' version of truth maintenance, an ATMS dependency network offers a full history of dependencies using an efficient labeling algorithm. It offers a history of everything that has happened contrasted to just the "here and now" as provided by Jess' *logical* keyword. Dependency tracking and proof histories have been researched [4, 11, 9] and implemented in other rule based expert system shells such as MIKE [8].

4.3 ATMS Background

Using de Kleer's model [4, 5, 6], an ATMS is composed of a set of nodes, $n_1, n_2, \ldots, n_n$, where each node is a propositional variable. A proposition represents either a premise, contradiction, or assumption. A premise is a node that is always true. A contradiction is a node that is always false. An assumption is a node whose values may be changed by the inference engine during rule firings. The inference engine incrementally transmits these propositions (i.e. nodes) to the ATMS.

When the inference engine fires a rule that results in a new or modified fact, a justification is transmitted to the ATMS. A justification is a tuple consisting of the rule's antecedent and consequent forming the inference. Suggesting an "if-then" type of implication, a justification may only contain positive literals and be represented as a horn clause.

An ATMS node has a datum, justification, and label associated with it. The datum represents a rule or fact within the inference engine. The justification is composed of an antecedent, consequent, and informant. The antecedent represents facts on the left hand side of the rule that caused the rule's premises to be true and resulted in an activation and firing. The consequent represents facts that were asserted on the right hand side of the rule upon firing. The informant describes the type of deduction and is never used in any ATMS computations. It may be supplied to the inference engine to provide textual cues for explanation generation.

4.3.1 Interfacing an ATMS to the Jess Rule Engine

Interfacing an ATMS to a production rule system has been previously investigated by Morgue and Chehire [18]. In their study, two levels of coupling were described with respect to the match-select-act cycle of an inference engine. When an ATMS is loosely coupled with an inference engine, the select and act steps are modified to enable integration. This is a simple form of interaction between the ATMS and inference engine and is more prone to becoming intractable than a tight coupling approach. In tight coupling, the match step is modified. This requires changes to the engine's Rete algorithm.

To extend LCCMA with full dependency tracking via an ATMS, Jess offers sophisticated event handling that will readily enable communication between the Jess inference engine and the ATMS. Event handlers will be supplied and invoked when, for example, a fact is asserted, retracted, or modified. In conjunction with an ATMS facility, these handlers could build and maintain a complete history and dependency network.

Inspired from the Lisp interface definition of Forbus and de Kleer [11], an object oriented model of an `Atms` class has been designed. Java method signatures for an `Atms` interface were developed and are analogous to the Lisp functions. The `Atms` class depends upon an `AtmsInterface` and will thus implement the interface's methods. An `InferenceEngine` class realizes the `AtmsInterface`. A `createNode` and `justifyNode` method of the `Atms` class are called after the `Atms` object receives a message from the inference engine indicating that a new fact was created or a fact was modified by a rule firing resulting in an `ACTIVATION` event.

5. CONCLUSION AND FUTURE WORK

An agent that monitors Space Shuttle ground telemetry data was presented. LCCMA provides an increased insight for NASA system and hardware engineers. LCCMA has successfully detected launch commit criteria warnings and violations on a simulated playback data stream. We are investigating extending this agent with truth maintenance capabilities to support advanced diagnostics and prognostics.

Other future work includes incorporating probabilities of occurrence of faults within support equipment. In terms of the ATMS, this translates into the probabilities of a fact being derivable and the context within which it would appear. Previous research has shown the utility of Bayesian networks and their applicability for constructing probability distributions from an ATMS [22].

Brachman and Levesque [2] propose description logics to implement a production system, act as the working memory, or provide some other service to such a system. In this paper, the subsumptive power of description logics might be leveraged by the label update algorithms of the truth maintenance system. Further, the agent and Jess itself are implemented in Java, an object oriented language. Description logic taxonomies might be constructed to naturally mirror the object oriented models of the agents.

6. REFERENCES

[1] L. Bölöni and D. C. Marinescu. An Object-Oriented Framework for Building Collaborative Network Agents. In H. Teodorescu, D. Mlynek, A. Kandel, and H.-J. Zimmerman, editors, *Intelligent Systems and Interfaces*, International Series in Intelligent Technologies, chapter 3, pages 31–64. Kluwer Publising House, 2000.

[2] R. Brachman and H. Levesque. *Knowledge representation and reasoning*. Morgan Kaufmann, May 2004.

[3] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley, Reading, MA, 1986.

[4] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, Mar. 1986.

[5] J. de Kleer. Extending the ATMS. *Artificial Intelligence*, 28(2):163–196, Mar. 1986.

[6] J. de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, 28(2):197–224, Mar. 1986.

[7] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, November 1979.

[8] M. Eisenstadt and M. Brayshaw. Build your own knowledge engineering toolkit. Technical report, Human Cognition Research Laboratory, The Open University, UK, June 1990.

[9] R. Filman. Reasoning with worlds and truth maintenance in a knowledge-based programming environment. *Communications of the ACM*, 31(4):382–401, Jan 3-6 1988.

[10] FIPA. Foundation for intelligent physical agents abstract architecture specification, Dec. 2002.

[11] K. D. Forbus and J. de Kleer. *Building Problem Solvers*. MIT Press, Cambridge, MA, 1993.

[12] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Artificial Intelligence*, volume 19(1), pages 17–37, 1982.

[13] E. Friedman-Hill. *Java Expert System Shell*. Manning Publications, Greenwich, CT, 2003.

[14] E. Gamma, R. Helm, E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Greenwich, CT, 1995.

[15] H. Gehman, S. Turcotte, J. Barry, K. Hess, J. Hallock, S. Wallace, D. Deal, S. Hubbard, R. Tetrault, S. Widnall, D. Osheroff, S. Ride, and J. Logsdon. *Columbia Accident Investigation Board (CAIB), Volume 1*. NASA, Washington D.C., August 2003.

[16] JADE. Java agent development framework. http://jade.tilab.com/, 2004.

[17] Lockheed. Pcgoal requirements document. Technical Report KSCL-1100-0804, Lockheed Space Operations Company, Oct. 1991.

[18] G. Morgue and T. Chehire. Efficiency of production systems when coupled with an assumption based truth maintenance system. In *Proc. of AAAI-91*, pages 268–274, Anaheim, CA, 1991.

[19] NASA. The vision for space exploration. Technical Report NP-2004-01-334-HQ, NASA, Feb 2004.

[20] NASA. The new age of exploration: Nasa's direction for 2005 and beyond. Technical Report NP-2005-01-397-HQ, NASA, Feb 2005.

[21] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[22] S. Srinivas. A probabilistic atms. Technical Report KSL 94-13, Knowledge Systems Laboratory, Stanford University, Feb. 1994.

[23] Sun. Java bean specification. http://java.sun.com/, 2004.

[24] M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press, Cambridge, Massachusetts, 2000.

[25] R. M. Wygant. Clips: A powerful development and delivery expert system. In *Computers and Industrial Engineering*, volume 17, pages 546–549, Anaheim, CA, 1989.

[26] D. E. Young. Lisa - intelligent software agents for common lisp. http://lisa.sourceforge.net, 2004.